

## Attached Appendix A

- **Host and Network Monitoring:**
  - Collection of Data via Operating System Calls for Minimal Intrusiveness
  - Common Data Formats Across Platforms
  - Distribution of Host and Network Statuses and Performance Histories
  - Discovery of Distributed Environment Configuration Changes
    - Detection of Host Failure
    - Detection of Host Startup
- **Application-Level Instrumentation:**
  - Low-Overhead Application API's
  - Common User-Specified Instrumentation Data Formats
  - Data Collection and Distribution Architecture
  - Grammar-Driven Event Correlation
- **System Specifications:**
  - Modeling of Application Systems
    - Structure, Capabilities, and Configuration
    - Requirements and Inter-Dependencies
  - Modeling of Hardware and Network Systems
    - Structure, Capabilities, and Configuration
  - Run-Time Access to Specification Information
    - Run-Time Loading of Specification Information
    - Run-Time Access via Object-Oriented API
- **Resource Allocation Decision-Making:**
  - Determination of Application-to-Host Mappings
  - Recovery from Hardware and Software Failures
  - Detection of and Recovery from Software Performance Problems
    - Control of Application Scalability
    - Reallocation of Applications to Hosts
  - Reallocation of Applications to Hosts based on Priority Changes
    - Application-to-Host Mappings for New Required Applications
    - Selection of Applications to be Shutdown
  - Resolution of Inter-Application Startup Dependencies
- **Resource Control:**
  - Startup, Shutdown, and Configuration of Distributed Applications
    - Interactive Operator Control via Operator Display
      - Creation of Defined System Configurations
      - Loading of Pre-defined System Configurations
    - Startup, Shutdown, and Configuration of Individual Applications
  - Automatic Control via Resource Manager Orders
  - Failure Detection Capabilities
    - Application Failure Detection via Interrupt Notification
    - Host Failure Detection via Internal Heartbeat Mechanism
- **Displays / Visualization:**
  - Host Configuration and Performance
  - Network Configuration and Performance
  - Application Software Performance
  - Resource Allocation Decisions and State Information
  - Software Status and Configuration
  - User-Configurable Instrumentation Display
  - Near Real-Time Display of Information
- **Middleware**
  - Reliable Message Passing
  - Location-Transparent TCP Client-Server Configuration
  - Automated Connections and Reconnections
    - Client and Server detection via UDP multicast
  - Many-to-Many Client-Server Connections supported
  - Message Callback Function Registration
  - TCP Connection Status Change Callback Function Registration

## Attached Appendix B:

### 1) Event data message header:

```
long total_bytes // total number of bytes in the event data message
long message_type // message type designator *
char version[8] // version of Instrumentation APIs *
char test_name[24] // test name for this event
double timetag // time stamp of when this event data message was sent
unsigned int gm_time // GMT time stamp
unsigned int event_num // event number
char process_name[24] // name of application sending event data message
long pid // process id of application sending event data message
char host_name[64] // host name that application is running on
long ip_addr // ip address of host
long tid // task id of application sending event data message
unsigned int thread_type // thread type
unsigned int sequence_num // sequence number of the event data message
double time_in_client // time the API was called to create event data message
double time_server_received // time the Instrumentation Daemon read in this event data message
double time_server_sent // time the Instrumentation Daemon sent event data message to the Instrumentation Collector
```

2) The event data message format string contains data field names and format specifiers for each data field. The following data specifiers (borrowed from ANSI C) are supported:

```
%*r : raw data, user defined, and number of bytes
%hi : short signed 16 bit integer
%hd : short signed 16 bit integer
%hu : short unsigned 16 bit integer
%li : long signed 32 bit integer
%ld : long signed 32 bit integer
%lu : long unsigned 32 bit integer
%lf : IEEE double precision floating point - signed 64 bit floating point
```

- %s : null-terminated string data
- %c : character
- %f : IEEE single precision floating point - signed 32 bit floating point
- %i : signed 32 bit integer
- %d : signed 32 bit integer
- %u : unsigned 32 bit integer

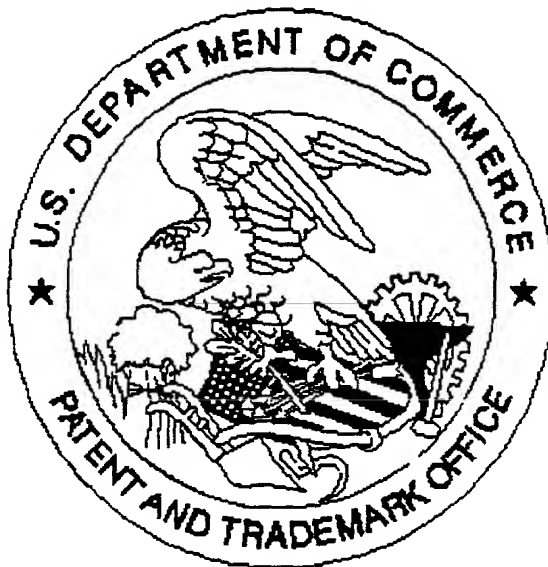
Event string example: "StartTime %.f StopTime %.f TrackNumber %u Hostname %s"

3) The data fields are then packed using the MessageBuffer class described in Appendix A of the RMComms Middleware Design Report.

## Attached Appendix C

TCPCCommClient	<p>RMComms client server communication client services:</p> <ul style="list-style-type: none"> <li>▪ client configuration <ul style="list-style-type: none"> <li>- client name, server port number, network interface to use (optional)</li> </ul> </li> <li>▪ connection and disconnection to servers <ul style="list-style-type: none"> <li>- all servers, specific servers, or servers on specific hosts</li> </ul> </li> <li>▪ sending user-defined messages to connected servers <ul style="list-style-type: none"> <li>- send to all servers or only to specific servers</li> </ul> </li> <li>▪ receiving user-defined messages from connected servers <ul style="list-style-type: none"> <li>- registration of message handler callback functions for specific messages</li> <li>- polled or asynchronous message delivery</li> </ul> </li> <li>▪ monitoring of server connection statuses <ul style="list-style-type: none"> <li>- queries to determine connected server statuses</li> <li>- notification of new server connections or broken server connections</li> </ul> </li> </ul>
TCPCCommServer	<p>RMComms client-server communication server services:</p> <ul style="list-style-type: none"> <li>▪ server configuration <ul style="list-style-type: none"> <li>- server name, server port number, network interface to use (optional)</li> </ul> </li> <li>▪ connection to new clients</li> <li>▪ sending user-defined messages to connected clients <ul style="list-style-type: none"> <li>- send to all clients or only to specific clients</li> </ul> </li> <li>▪ receiving user-defined messages from connected clients <ul style="list-style-type: none"> <li>- registration of message handler callback functions for specific messages</li> <li>- polled or asynchronous message delivery</li> </ul> </li> <li>▪ monitoring of client connection statuses <ul style="list-style-type: none"> <li>- queries to determine connected client statuses</li> <li>- notification of new client connections or broken client connections</li> </ul> </li> </ul>
TimeUtils	<p>Clock access and time conversion services:</p> <ul style="list-style-type: none"> <li>▪ read system clock time</li> <li>▪ time conversions between GMT and local time</li> <li>▪ time conversions to hours, minutes, seconds, day, month, year</li> </ul>
SignalRegistry	<p>User-defined signal (interrupt) handler registration services:</p> <ul style="list-style-type: none"> <li>▪ register a signal handler function for a specified signal <ul style="list-style-type: none"> <li>- invoked when interrupt occurs</li> </ul> </li> <li>▪ unregister a signal handler function for a specified signal</li> </ul>

United States Patent & Trademark Office  
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☐ Page(s) \_\_\_\_\_ of \_\_\_\_\_ were not present  
for scanning. (Document title)

☐ Page(s) \_\_\_\_\_ of \_\_\_\_\_ were not present  
for scanning. (Document title)

☒ Scanned copy is best available. Drawings -